Esqueletização de uma Imagem

Paradigmas de Computação Paralela

Mestrado Integrado em Engenharia Informática

Universidade do Minho - Departamento de Informática

Bruno Miguel da Silva Barbosa
Emanuel Queiroga Amorim Fernandes
{a67646,a61003}@alunos.uminho.pt

Resumo

Neste relatório é apresentada a análise de um algoritmo de esqueletização de imagem no âmbito do paradigma de programação baseado em memória partilhada recorrendo à API do OpenMP. A abordagem do tema começa pela implementação sequencial do algoritmo. Segue-se posteriormente, a versão paralela do mesmo juntamente com a comparação dos diversos tempos de execução e anális de resultados. Por último, apresentamos uma conclusão referente à distinção entre estas duas metodologias de programação e a indicação dos dados de entrada e as otimizações realizadas. Nos anexos encontram-se as imagens resultantes (esqueletos) durante toda a experimentação.

1. Introdução

A esqueletização de uma imagem é uma das inúmeras operações de processamento de imagem. O algoritmo usado neste trabalho procede varrendo todos os pixeis da imagem e faz cálculos com os pixeis adjacentes. Portanto, percebemos instantaneamente, que se tivermos uma imagem com grande resolução, que é bastante comum nas câmaras fotográficas de hoje em dia, a execução do algoritmo pode ser muito ineficiente. No nosso caso utilizamos um tipo específico de imagens, com a extensão .pbm que nos poupa bastante trabalho durante o parsing do ficheiro. A sua estrutura é muito simples: na primeira linha tem um indicativo do tipo de ficheiro, normalmente P1; na segunda linha tem-se o número de colunas e o número de linhas; e daí em diante, temos os valores respetivos a cada pixel. Para além da sua sintaxe ser de fácil compreensão, trata-se de uma imagem binária, pois os pixeis possuem valor 0 ou 1, ou seja, branco ou preto, respetivamente.

Posto isto, hoje em dia existem muitas aplicações dependentes destes tipos de algoritmos. A paralelização torna-se assim fundamental na medida em que acelera o processamento dos vários pixeis, tornando-o desta forma mais eficiente, o que é bom quando toca a aplicações onde são necessários resultados em tempo real.

Assim sendo, nosso trabalho visa não só a paralelização do algoritmo mas também a sua otimização, com o objectivo de tirar o máximo partido dele no menor tempo possível.

2. Versão Sequencial

2.1. Especificação do algoritmo

A primeira implementação do algoritmo de esqueletização é a versão sequencial. Como já foi referido anteriormente, as operações são feitas pixel a pixel. São feitas duas passagens pela imagem onde a cada iteração é realizado um conjunto de condições que determinam se um pixel devera pertencer, ou não, ao esqueleto da

imagem. Para se remover o pixel todas as condições têm de ser respeitadas. Para um dado pixel, que não pertença à borda da imagem, temos a representação que se segue:

p9	p2	p3
p8	p1	p4
p7	p6	p5

A nível de pseudocódigo o algoritmo sequencial é descrito da seguinte maneira:

Repetir {

1ª Passagem - Remover p1 se {

(i) O número de vizinhos com valor 1 está compreendido entre 2 e 6.

(ii) O número de transições 0-1 é igual a 1.

(iii) $\underline{p4} + \underline{p6} + \underline{p2p8} = 1$

}

2ª Passagem - Remover p1 se {

(i) O número de vizinhos a 1 está compreendido entre 2 e 6.

(ii) O número de transições 0-1 é igual a 1.

(iii) $\underline{p2} + \underline{p8} + \underline{p4p6} = 1$

}

} Até não remover mais pixeis

Nota: Por exemplo, $\underline{p1}$ significa o complemento de p1.

3. Versão Paralela

3.1 Análise do algoritmo

Numa primeira análise do algoritmo de esqueletização, observamos que este possui um ciclo que por sua vez contem duas passagens por uma matriz, cada uma destas inclui dois ciclos 'for', terminando o ciclo quando não é possível alterar mais a matriz, (alterar o valor de uma célula com o valor 1 por 0). Desta parte do algoritmo, devido ao ciclos 'for' tentamos paralelizar o código com a diretiva '#pragma omp for ordered', onde não obtivemos resultados satisfatórios. O grupo acha que os resultados insatisfatórios desta primeira tentativa devem-se à natureza sequencial do algoritmo, onde cada iteração do ciclo 'for' onde é alterada uma célula da matriz, tem hipóteses de alterar o comportamento da iteração seguinte, portanto o custo de criar as threads e/ou reutilização delas, acaba por prejudicar o desempenho visto que a thread que irá trabalhar na iteração seguinte à atual tem de esperar que a anterior termine a sua execução, exceptuando os casos em que as threads possuem uma célula a '0', mas nesse caso quase não há trabalho a executar.

Olhando para dentro dos ciclos 'for' o mesmo problema aparece, existem três condições a ser verificadas, mas cada condição só é necessário ser verificada caso a precedente se verificar, o que induz um procedimento sequencial ou executar tarefas que poderão ser desnecessárias, a segunda hipótese seria demasiado influenciada pelo *input*, motivo que levou então o grupo a não criar tarefas a executar em paralelo para cada condição.

3.2 A Solução alcançada

A versão paralela do algoritmo surge, desta forma, dum exame mais aprofundado da versão sequencial. Identificamos que a melhor alternativa para paralelizar o algoritmo, seria segmentar horizontalmente a imagem pelo número de threads, onde cada segmento é executado em sequencial, expectando um ganho significativo em relação à versão sequencial.

Deste processo de paralelização criamos duas versões paralelas às quais intitulamos skel_v2 e skel_v3, vamos falar agora das principais características/diferenças das duas versões.

- skel_v2: Nesta versão usamos a diretiva *pragma omp for* com o *schedule static* e o *chunk_size* igual ao número de linhas menos duas (o algoritmo não considera as bordas) a dividir pelo numero de *threads*. Usamos esta técnica tanto no *loop* da primeira passagem como na segunda. Para obter resultados satisfatórios ao nível de *speed-ups* não tratamos das dependências no interior dos loops, como por exemplo, a ordem sequencial entre as threads necessária para uma correta execução. Um dos problemas desta implementação é a possível diferença de *output* gerado em diferentes execuções do programa para o mesmo *input* (diferentes velocidades na execução das *threads* podem implicar leitura de valores diferentes nos limites das linhas tratadas pelas *thread*s . O grupo observou que essa diferença aumenta com o numero de *thread*s usadas (para duas *thread*s a diferença é menor que 1% enquanto que para 12 *threds* poderia chegar aos 20%)
- skel_v3: Nesta versão foram criadas *tasks* chamadas por uma única *thread*, que executa um ciclo for que itera secções horizontais da matriz, ou seja cada *thread* da *team* executa uma só *task* que contem a secção que lhe corresponde. Nesta versão dividimos melhor a imagem criando uma 'borda' o que retira as dependências entre as *threads*, cada uma trata a sua parte da imagem sem influenciar as outras, o que faz com que diferentes execuções do programa produzam o mesmo *output*. No final deste processo a matriz é de novo processada, agora de forma sequencial, numa tentativa de 'remediar' algumas diferenças causadas por este método.
- Em ambas as versões o segundo varrimento da matriz apenas é efetuado quando o primeiro termina, respeitando a ordem do algoritmo.

Da mesma forma, também identificamos que a paralelização pode por em causa a qualidade dos resultados quando são desrespeitadas as dependências de dados, isto é, quando o resultado de uma operação influencia a próxima operação. Por exemplo, na primeira alternativa, os pixeis que estão localizados na transição de um segmento para o outro são susceptíveis de ficarem mal calculados devido a certas dependências de dados que não sejam respeitadas.

4. Resultados obtidos

Nesta secção apresentamos, sumariamente, os resultados obtidos durante todo este processo de experimentação. Os resultados não se limitam a tempos de execução porque também é importante avaliar a qualidade dos resultados provenientes da versão paralela (que devem o mais idênticos possíveis, se não iguais, aos da versão sequencial). A nossa abordagem vai ter em conta que a diferença entre o resultado obtido, neste caso a imagem resultante, na versão sequencial e paralela.

Contudo, alguns dos resultados obtidos a nível de tempos de execução já eram esperados, nomeadamente aqueles onde não se alcança um speedup proporcional ao número de threads. Por motivos de simplicidade, restringimos no máximo, a execução com 12 threads.

4.1 Análise de performance

A Tabela 1 apresenta os resultados obtidos no geral, recorrendo um único nodo (641), tanto na versão sequencial como nas duas versões paralelas. Os tempos de execução estão medidos em segundos, sendo o valor apresentado a mediana de 5 medições. Os cálculos da diferença entre o esqueleto da versão sequencial e paralelo é determinado graças a um software chamado Image Magic. Na linha de comandos efetuamos a sua chamada da seguinte forma:

compare -metric RMSE imagem1 imagem2 NULL

O valor da percentagem é o que surge entre parêntesis.

Número de Threads	Tempo Base Sequencial	Tempo Paralelo (v2)	Tempo Paralelo (v3)	Speedup (v2)	Speedup (v3)	Diferença % (v2)	Diferença % (v3)
1	15.059823	15.078925	15.152625	0.99	0.99	0	0
2	15.059823	7.813454	9.572198	1.93	1.57	14.21	13.91
4	15.059823	5.548109	5.50241	2.71	2.74	25.18	24.67
8	15.059823	2.75117	3.86600	5.47	3.90	33.01	31.07
12	15.059823	2.34167	3.098033	6.43	4.86	35.02	32.64

Tabela 1. Comparação dos tempos de execução com a imagem Seahorse.

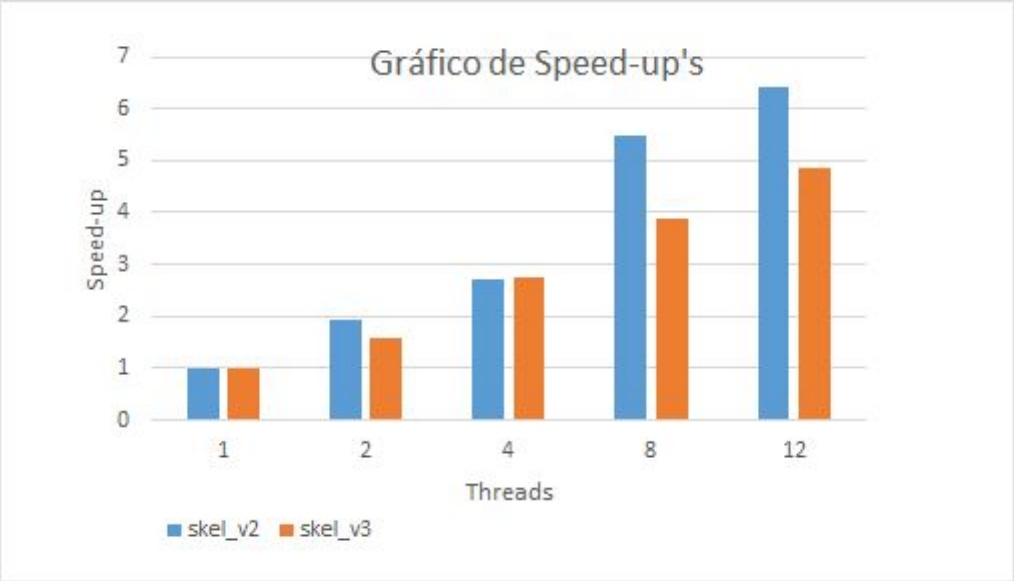

Imagem 1. Gráfico de speedup relativo aos resultados da tabela 1.

4.2 Análise de resultados

Analisando os dados obtidos podemos verificar que obtivemos bastantes bons resultados em termos de speed-up principalmente para um número de *threads* inferior a oito, perdendo-se escalabilidade quanto maior é o número de *threads*. Onde se pode verificar que mesmo dividindo a imagem em blocos por número de *threads* o *speed-up* não iguala esta mesma divisão.

A tabela 1 ilustra bem a perda de precisão dos resultados obtidos quando se aumenta o número de *threads*, o que já era espectável, quanto mais dividida a imagem mais diverge o algoritmo paralelo do sequencial. Se as diferenças são aceitáveis ou não depende de quem irá usar o algoritmo, no entanto mesmo uma imagem gerada por o algoritmo com 12 *threads* constrói um esqueleto global bastante semelhante.

Alguns dos motivos que podem levar à gradual perda de escalabilidade aquando o aumento de *threads* são: O custo adicional de criação de novas *threads*, apesar do reaproveitamento do *openmp* de threads antigas este custo é elevado no nosso algoritmo devido à grande quantidade de variáveis privadas para cada fio de execução. Outro motivo poderá ser as diferenças causadas pelo não respeito da ordem sequencia nos limites das

linhas da matriz que cada fio de execução processa, que acresce com o numero de *threads*. As partes sequenciais do código também introduzem um peso significativo nesta falta de escalabilidade, um grande fator deste tipo é a passagem sequencial pela matriz na versão skel_v3 após a paralela ter terminado. A largura de banda da memoria também se torna um problema quando aumenta o numero de threads, devido a grande parte do algoritmo implicar pesquisa e alteração de uma matriz.

5. Dados de entrada

Para a obtenção destes resultados usamos apenas uma imagem (seahorse.ascii.pbm) com uma resolução de 2225 por 3601 pixeis. A principal razão desta escolha deve-se ao facto de ter um tamanho relativamente razoável, ao contrário de todas as outras que conseguimos encontrar. As restantes são muito mais pequenas e à medida que aumentamos a paralelização os erros tornavam-se cada vez mais evidentes. Todavia, a nível de testes locais optamos por uma imagem mais compacta (washington.ascii.pbm), com uma resolução de 305 por 400 pixeis, de modo a conseguir resultados em menor tempo na ordem das décimas de segundo.

6. Otimizações

Relativamente a otimizações, apenas realizamos uma. Já na versão sequencial reduzimos os acessos a memória de 30 para 8 acessos por iteração, através da declaração de variáveis locais para os pixeis adjacentes ao pixel atual. Desde então as versões paralelas derivaram dessa. Não realizamos, no entanto, testes que comprovassem a melhoria mas reconhecemos que se trata de um ganho significativo devido à latência da memória.

Anexos

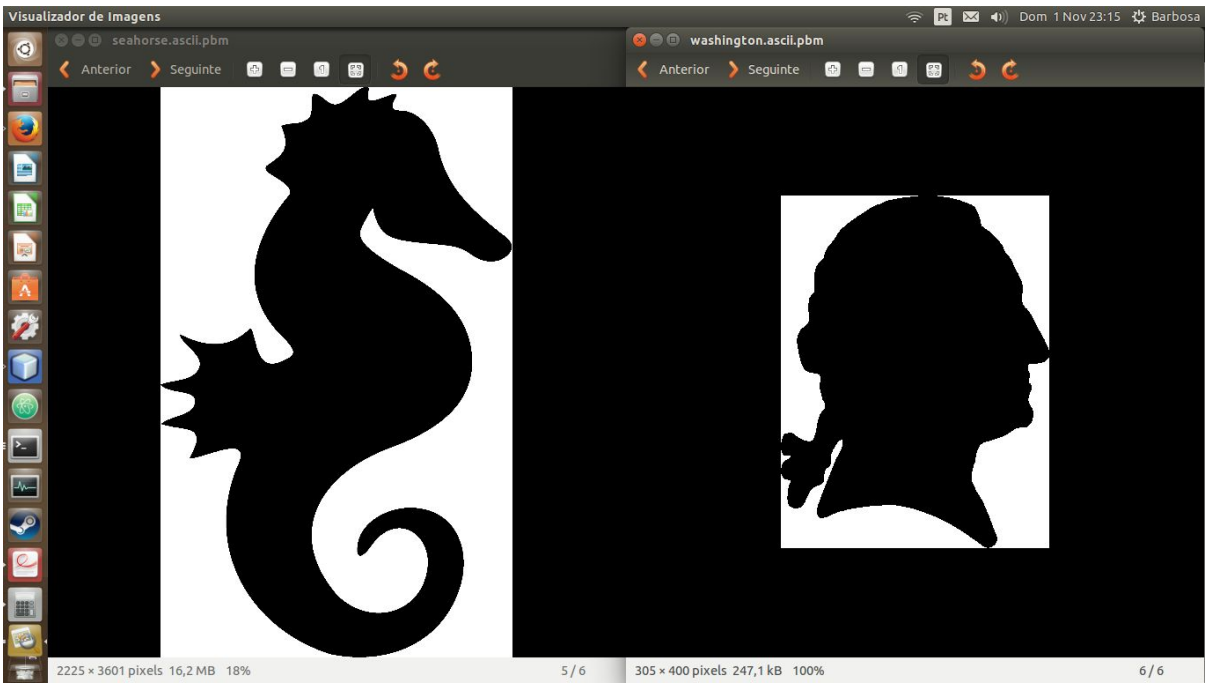

Imagem 2. Imagens usadas (seahorse e washington, respetivamente) como dados de entrada.

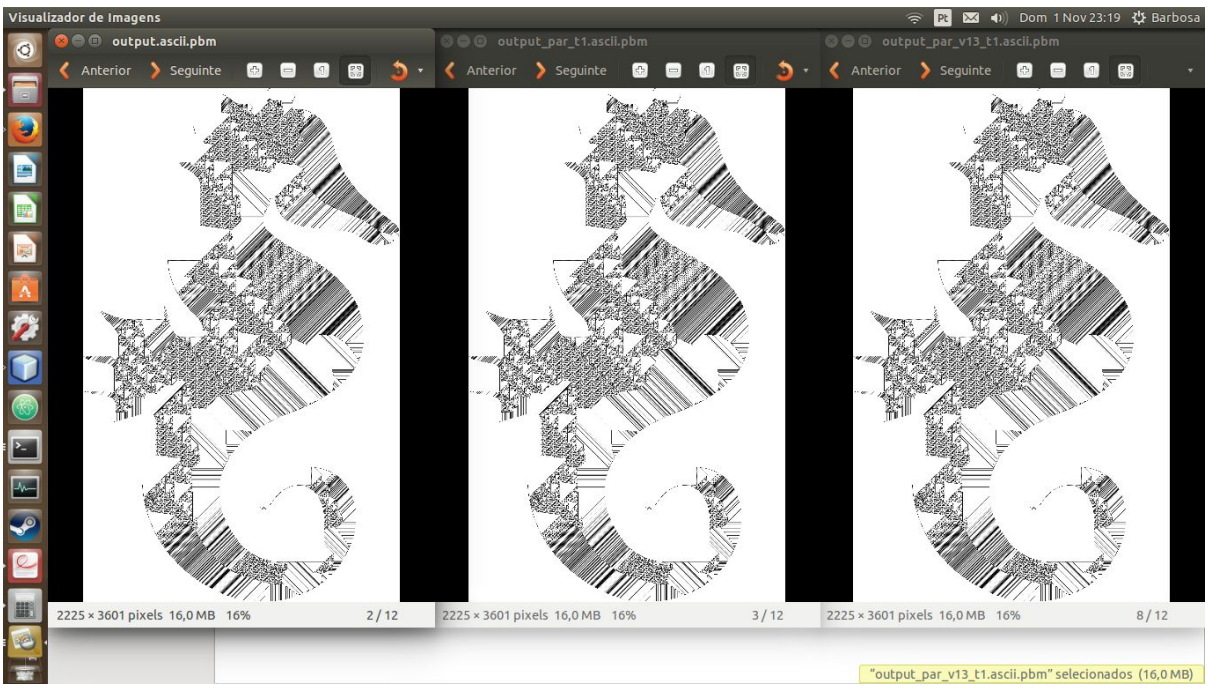

Imagem 3. Comparação entre sequencial, v2 para 1 thread e v3 para 1 thread (seahorse).

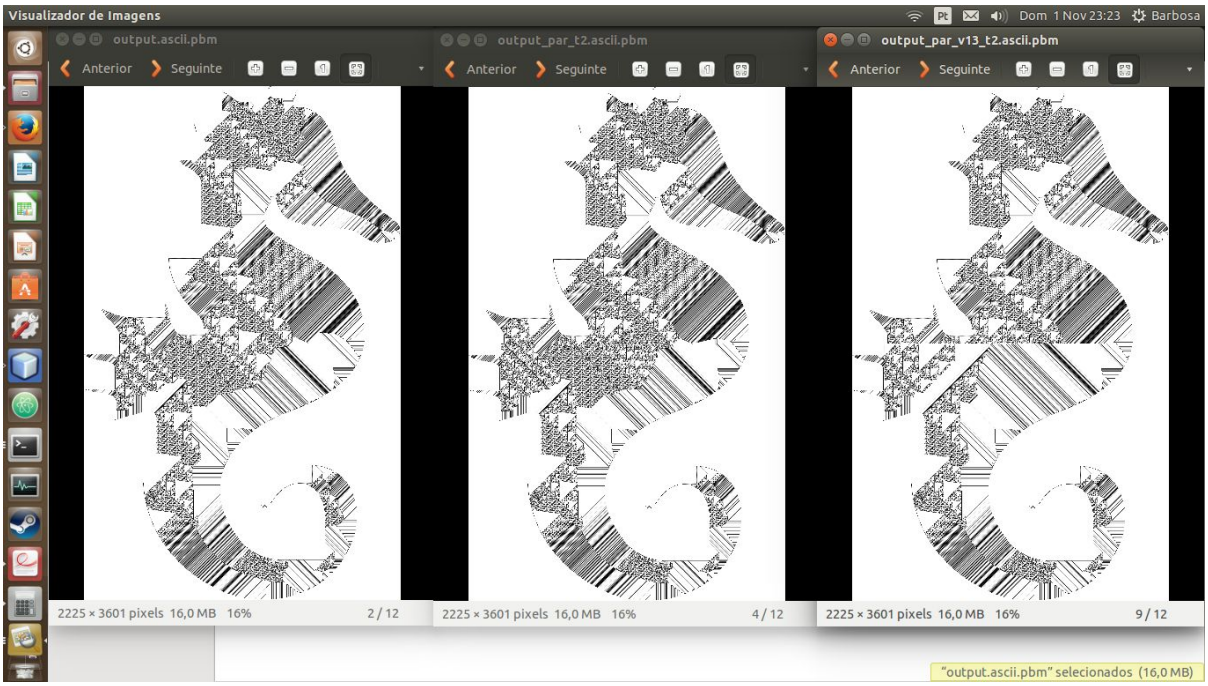

Imagem 4. Comparação sequencial, v2 com 2 threads e v3 com 2 threads (seahorse).

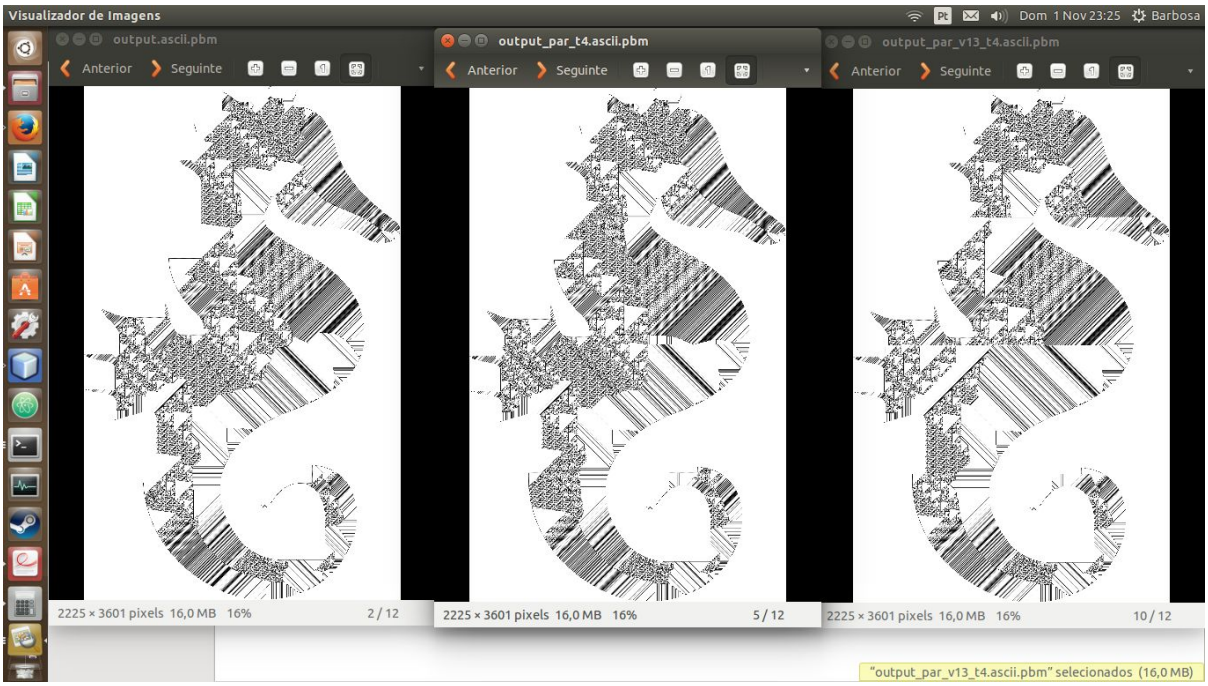

Imagem 5. Comparação entre sequencial, v2 com 4 threads e v3 com 4 threads (seahorse).

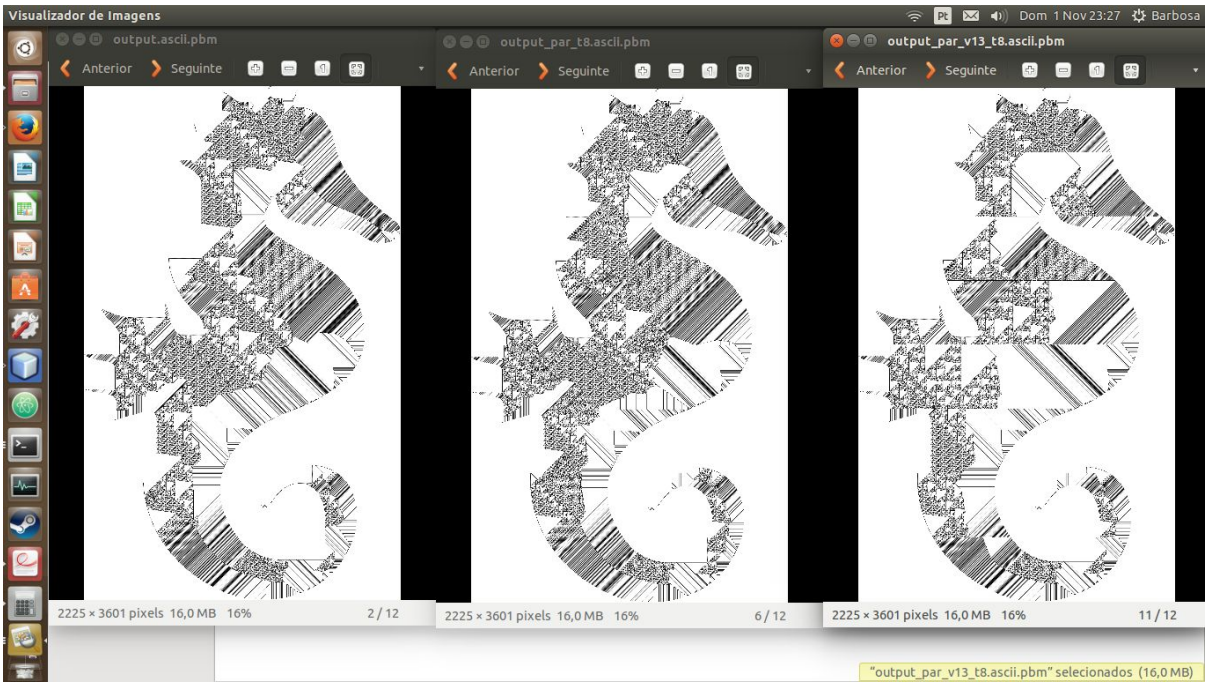

Imagem 6. Comparação entre sequencial e versões paralelas com 8 threads (seahorse).

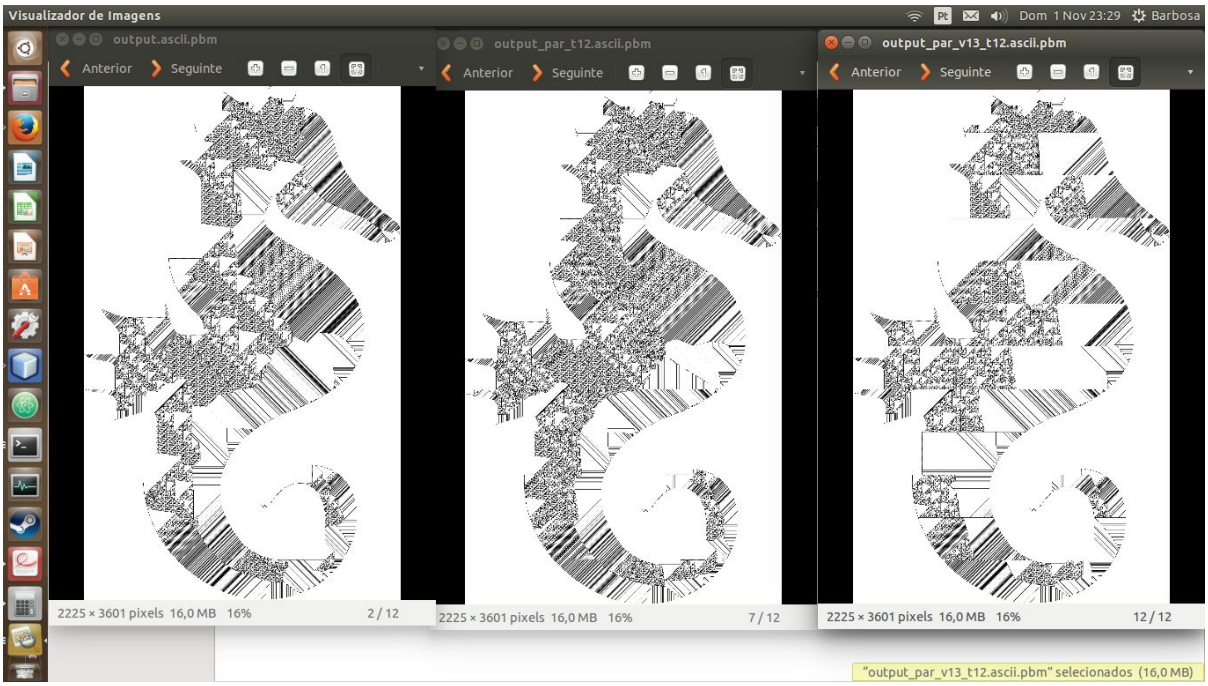

Imagem 7. Comparação entre sequencial e versões paralelas com 12 threads (seahorse).

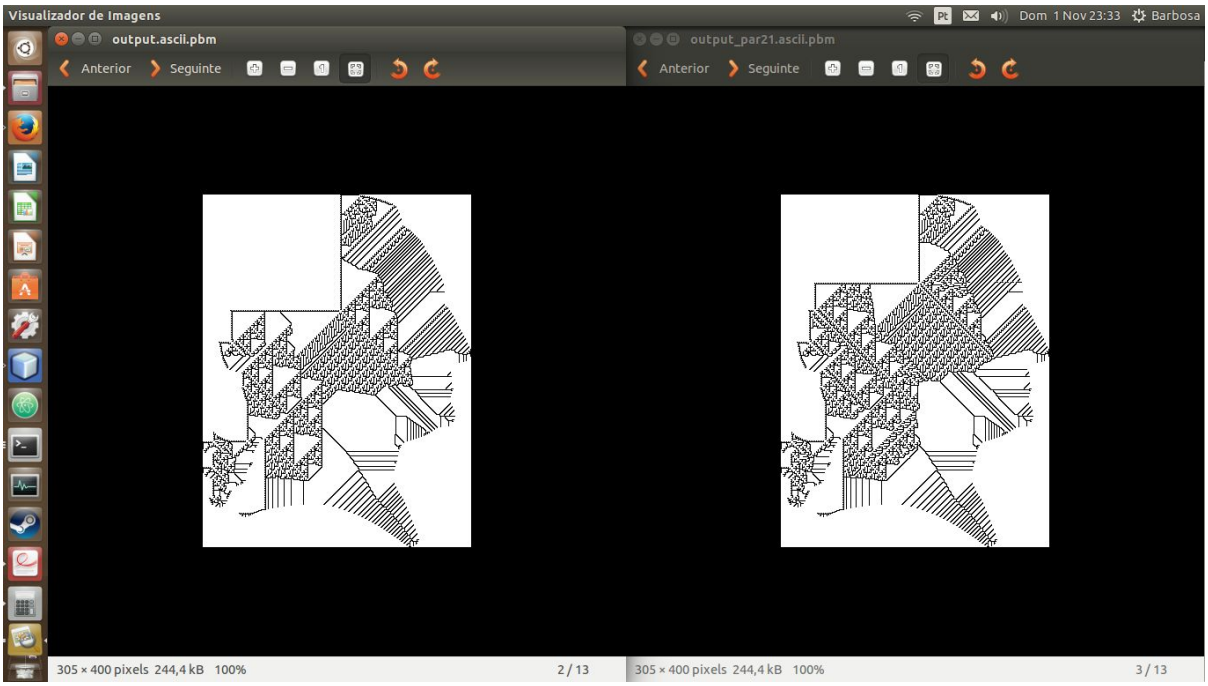

Imagem 8. Esqueletização sequencial e paralela da imagem washington.

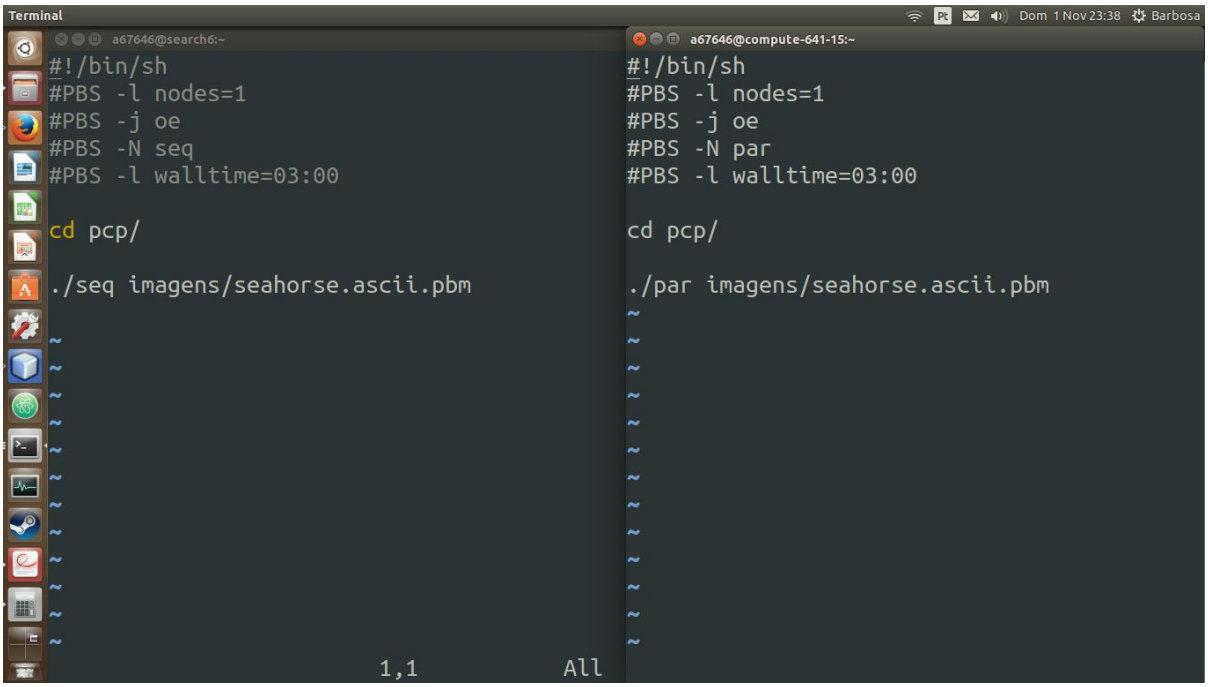

Imagem 9. Ficheiros .scr usados para submissão de trabalhos no cluster.

				skel_v2						skel_v3		
	sequencial	uma thread	duas threads	quatro threads	oito threads	doze threads		uma thread	duas thread	quatro threads	oito threads	doze threads
	15,558287	14,940481	7,834929	5,600844	2,75117	1,972487		15,077297	9,36512	5,468521	3,841581	3,009562
	14,109451	15,078925	7,740728	5,555683	2,765148	2,34167		15,104031	10,089987	5,785272	3,930817	3,060944
	14,95341	15,178482	7,754177	5,548109	2,77375	2,074736		15,978039	9,635687	5,627773	3,866	3,098033
	15,059823	14,92305	7,813454	5,427255	2,724897	2,88497		15,152625	9,572198	5,335452	3,866817	3,156831
	15,605968	16,086347	7,814987	5,343417	2,699453	2,353018		15,390054	9,196417	5,50241	3,810092	3,173297
mediana	15,059823	15,078925	7,813454	5,548109	2,75117	2,34167		15,152625	9,572198	5,50241	3,866	3,098033
	15,059823	15,059823	15,059823	15,059823	15,059823	15,059823		15,059823	15,059823	15,059823	15,059823	15,059823
speed_up	1	0,9987332	1,927421983	2,714406476	5,473970347	6,431231984		0,993875517	1,57328787	2,736950354	3,89545344	4,861091861

Imagem 10. Cálculos efetuados sobre os dados retirados do cluster para preenchimento da tabela 1.